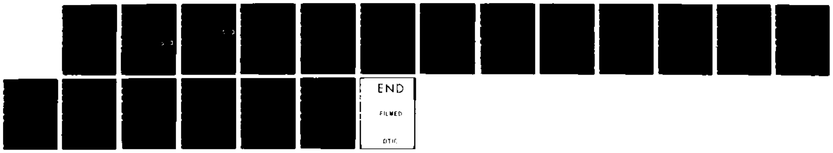
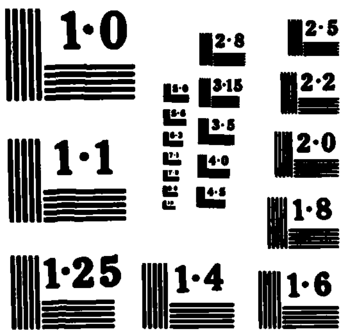

END
FILMED
DTIC


1·0
2·8
2·5
3·15
2·2
3·5
2·0
1·1
4·0
4·5
1·8
1·25
1·4
1·6

CAR-TR-70
CS-TR-1416

F49620-83-C-0082
July 1984

# THE PRISM MACHINE: AN ALTERNATIVE TO THE PYRAMID

Azriel Rosenfeld

Center for Automation Research
University of Maryland
College Park, MD 20742

COMPUTER VISION LABORATORY

CENTER FOR AUTOMATION RESEARCH

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

CAR-TR-70 F49620-83-C-0082
CS-TR-1416 July 1984

# THE PRISM MACHINE: AN ALTERNATIVE TO THE PYRAMID

Azriel Rosenfeld

Center for Automation Research
University of Maryland
College Park, MD 20742

## ABSTRACT

The prism machine is a stack of n cellular arrays, each of size $2^n \times 2^n$. Cell (i,j) on level k is connected to cells (i,j), $(i+2^k,j)$, and $(i,j+2^k)$ on level k+1, $1 \le k < n$, where the sums are modulo $2^n$. Such a machine can perform various operations (e.g., "Gaussian" convolutions or least-squares polynomial fits) on image neighborhoods of power-of-2 sizes in every position in O(n) time, unlike a pyramid machine which can do this only in sampled positions. It can also compute the discrete Fourier transform in O(n) time. It consists of $n \cdot 4^n$ cells, while a pyramid consists of fewer than $4^{n+1}/3$ cells; but in practice n would be at most 10, so that a prism would be at most about seven times as large as a pyramid.

Research sponsored by the Air Force Office of Scientific Research (AFSC), under Contract F49620-83-C-0082. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The help of Janet Salzman in preparing this report is gratefully appreciated.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

## REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release; distribution unlimited. |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S)<br>AFOSR-TR- 84-0830 |
| 6a. NAME OF PERFORMING ORGANIZATION<br>University of Maryland | 6b. OFFICE SYMBOL *(If applicable)* |
| 7a. NAME OF MONITORING ORGANIZATION<br>Air Force Office of Scientific Research | |
| 6c. ADDRESS (City, State, and ZIP Code)<br>Center for Automation Research<br>College Park MD 20742 | 7b. ADDRESS (City, State, and ZIP Code)<br>Directorate of Mathematical & Information Sciences, AFOSR, Bolling AFB DC 20332 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION<br>AFOSR | 8b. OFFICE SYMBOL *(If applicable)*<br>NM |
| 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER<br>F49620-83-C-0082 | |
| 8c. ADDRESS (City, State, and ZIP Code)<br>Bolling AFB DC 20332 | 10. SOURCE OF FUNDING NUMBERS |

| PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT ACCESSION NO. |
|---|---|---|---|
| 61102F | 2304 | A7 | |

11. TITLE *(Include Security Classification)*
THE PRISM MACHINE: AN ALTERNATIVE TO THE PYRAMID.

12. PERSONAL AUTHOR(S)
Azriel Rosenfeld

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Technical | FROM ____ TO ____ | JUL 84 | 16 |

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

| FIELD | GROUP | SUB-GROUP |
|---|---|---|
| | | |
| | | |

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Parallel processing; cellular arrays; pyramids; prisms.
X 2 to the n power x 2 to the n power

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The prism machine is a stack of n cellular arrays, each of size $2^n \times 2^n$. Cell (i,j) on level k is connected to cells (i,j), $(i+2^k,j)$, and $(i,j+2^k)$ on level k+1, $1 \le k < n$, where the sums are modulo $2^n$. Such a machine can perform various operations (e.g., "Gaussian" convolutions or least-squares polynomial fits) on image neighborhoods of power-of-2 sizes in every position in O(n) time, unlike a pyramid machine which can do this only in sampled positions. It can also compute the discrete Fourier transform in O(n) time. It consists of $n \cdot 4^n$ cells, while a pyramid consists of fewer than $4^{n+1}/3$ cells; but in practice n would be at most 10, so that a prism would be at most about seven times as large as a pyramid.

1 < or =

| | |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT<br>[✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION<br>UNCLASSIFIED |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>Dr. Robert N. Buchal | 22b. TELEPHONE (Include Area Code)<br>(202) 767- 4939 |
| 22c. OFFICE SYMBOL<br>NM | |

DD FORM 1473, 84 MAR    83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

84 09 17 005

## 1. Introduction

There has been considerable recent interest in a class of cellular computers known as "pyramids", which seem to be well suited for many types of image processing and analysis tasks. Pyramid architectures are especially appropriate when one wants to represent image information at a number of different scales. Two recent collections of papers on such multiscale approaches are [1] and [2].

Basically, a pyramid is a stack of cellular arrays that taper exponentially in size. For example, if the base of the pyramid is $2^n \times 2^n$, then the successively higher levels might be $2^{n-1} \times 2^{n-1}$, $2^{n-2} \times 2^{n-2}$, ..., $2 \times 2$, $1 \times 1$. Note that the total number of cells in such a pyramid is $4^n(1+\frac{1}{4}+\frac{1}{16}+\ldots)<4^{n+1}/3$, i.e., less than a third more cells than in the base alone. Each cell is connected to its neighbors on its own level, and also to a bounded number of cells on the levels above and below; for example, cell $(i,j)$ on level k might be connected to a "father" cell $(\lfloor\frac{i}{2}\rfloor,\lfloor\frac{j}{2}\rfloor)$ on level k+1, which implies that it is also connected to the "son" cells $(2i,2j),(2i+1,2j),(2i,2j+1)$, and $(2i+1,2j+1)$ on level k-1.

Pyramids can perform many types of operations on a $2^n \times 2^n$ image in $O(n)$ time. For example, the histogram of the image can be computed in $O(n)$ time by having each cell obtain counts of values from its sons and add them. Weighted averages of various types (e.g., having approximately Gaussian weights), with kernels of sizes $O(2^k)$, can be computed in positions spaced $O(2^k)$ apart by iterating small weighted-average computations in a pyramid [3]. Least-squares polynomial fits to (nonoverlapping) image blocks of sizes $2^k \times 2^k$ can be computed iteratively from the fits to blocks of sizes $2^{k-1} \times 2^{k-1}$ [4]. Polynomial fits to edges or curves in an image can also be computed iteratively [5].

This paper proposes an alternative to the pyramid which we call the "prism". A prism is a stack of cellular arrays, all of the same size. The height of the stack is the same as in the case of a pyramid; for example, if the base is $2^n \times 2^n$, the number of levels is n, so that the total number of cells is $n \cdot 4^n$. (Note that in practice n would be at most 10, so that a prism with base 1024×1024 would have only about 7 times as many cells ($10 \approx 7 \times \frac{4}{3}$) as a pyramid with the same base size.) Each cell is connected to its neighbors on its own level, and also to three cells on the level above; specifically, cell (i,j) on level k is connected to cells $(i,j)$, $(i+2^k,j)$, and $(i,j+2^k)$ on level k+1 (which implies that it is also connected to cells $(i,j)$, $(i-2^{k-1},j)$, and $(i,j-2^{k-1})$ on level k-1). In these connections, it is understood that addition is modulo $2^n$ - e.g., if $i+2^k > 2^n$, it means $i+2^k-2^n$. We number the levels 1,...,n, starting at the base. We will regard (i,j) as Cartesian coordinates.

We shall show that a prism can perform many of the same types of operations as a pyramid machine in O(n) time, but can perform them at every position in the image rather than at positions spaced $2^k$ apart. We shall also show that it can compute the discrete Fourier transform of a $2^n$ by $2^n$ image in O(n) time. On the other hand a prism does not represent the information in an image at a reduced scale; it can condense the information into a small number of cells, but these cells are spaced far apart. Thus it cannot simulate a pyramid in detecting basic region shapes such as blobs and ribbons using local operations, though it may be able to detect these shapes in other ways.

## 2. Condensation of information

In this section we describe how a prism machine can compute many useful types of information about the $2^k \times 2^k$ blocks of pixels in an image, for $k=1,\ldots,n$ and for blocks in every possible position, in time $O(n)$.

### 2.1. Histogramming

As a first and very simple example, suppose we want to count how many times a given gray level $z$ occurs in each block. (Doing this for each $z$ gives us the gray level histogram of the block.) At the bottom level of the prism, we trivially have this count for the $1\times1$ blocks: the count is 1 if the given pixel has value $z$, and 0 otherwise. Working within the bottom level, we add the count of each pixel's left neighbor to its own count; this yields, at each pixel $(i,j)$, the count for the $1\times2$ block of pixels $\{(i-1,j),(i,j)\}$. We now add this count for each pixel's lower neighbor to its own count; this yields, at each $(i,j)$, the count for the $2\times2$ block of pixels $\{(i-1,j),(i,j),(i-1,j-1),(i,j-1)\}$.

Next, for every $(i,j)$, we pass the counts from pixels $(i-2,j)$ and $(i,j)$ on the first level up to pixel $(i,j)$ on the second level and add them. This means that each pixel $(i,j)$ on the second level now has the count for a $2\times4$ block of original pixels. We pass this information back down to pixel $(i,j)$ on the first level. Now, for every $(i,j)$, we pass these counts from pixels $(i,j-2)$ and $(i,j)$ on the first level up to pixel $(i,j)$ on the second level and add them. Each pixel $(i,j)$ on the second level now has the count for a $4\times4$ block of original pixels - namely, the $4\times4$ block with $(i,j)$ in its upper right corner.

We now perform an analogous process between levels 2 and 3. The $4\times4$ block counts from pixels $(i-4,j)$ and $(i,j)$ on level 2 are passed up to pixel $(i,j)$ on level 3 and added, giving it the count for a $4\times8$ block. The results are passed back down to level 2, and these $4\times8$ block counts at pixels $(i,j-4)$ and $(i,j)$ are then passed back up to pixel $(i,j)$ on level 3 and added, giving it the count for an $8\times8$ block. Continuing in this way, we obtain counts for $2^k$ by $2^k$ blocks on level $k$, $k=1,2,\ldots$ Note that the positions of all blocks are defined modulo $2^n$; thus on level $n$, every pixel obtains the count for a $2^n\times2^n$ block - i.e., for the entire image.

The entire counting process takes two steps (of shifting and adding counts) within the bottom level, plus three steps (adding, passing back down, and adding again) between each successive pair of levels, yielding as the total number of steps $3n-1=O(n)$. We could reduce the number of steps to $2n$ if we doubled the height of the stack, and connected pixel $(i,j)$ on level $2k-1$ to pixel $(i+2^k,j)$ on level $2k$, and pixel $(i,j)$ on level $2k$ to pixel $(i,j+2^k)$ on level $2k+1$, for $k=1,2,\ldots,n$; this requires a stack of $2n+1$ levels, but it reduces the number of connections between levels, and in the counting algorithm it eliminates the need to pass information back down. Note also that our algorithm does not use any within-level connections except at level 1. We could eliminate even the level-1 connections if we added a level 0, where pixel $(i,j)$ on level 0 is connected to pixels $(i,j),(i+2^0,j)=(i+1,j)$, and $(i,j+2^0)=(i,j+1)$ on level 1. However, the within-level connections are useful in performing local operations on the image (though we could simulate them by passing information up and down, if desired, at the cost of a constant factor in computation time).

## 2.2. Other blockwise operations

The method used to compute block counts can be trivially modified to compute block sums, maxima, etc. Level k obtains sums (etc.) for pairs of blocks of size $2^{k-1}\times 2^{k-1}$ from level k-1, adds them to obtain results for blocks of size $2^{k-1}\times 2^{k}$, passes the results back down and again adds pairs of them to obtain results for blocks of size $2^{k}\times 2^{k}$.

Computing iterated convolutions (i.e., _weighted_ sums) is slightly more complicated, since in general different weights must be applied to the pixels in a block. The simplest way to perform a 2×2 convolution is for each cell to obtain its left neighbor's value and then pass both values (its own and the neighbor's) to its upper neighbor; this gives each cell the quadruple of values in the 2×2 block, and it can now compute their weighted sum. To iterate the convolution, each cell (i,j) on level 2 receives a pair of values from cells (i-2,j) and (i,j) on level 1, passes them back down to level 1, and then receives two pairs of values from cells (i,j-2) and (i,j) on level 1; each of these four values represents the weighted sum of a 2×2 block. The cell can then compute the weighted sum of these four values, thus iterating the convolution; and so on at higher levels. To perform an iterated 4×4 convolution, each cell (i,j) on level 1 collects all the values in the 2×2 block without combining them, and each cell (i,j) on level 2 then collects these 4-tuples of values, thus giving it all the values in a 4×4 block; it then computes the weighted sum of these 16 values. The process is then iterated by passing data up two more levels before combining it; and so on. Modifications of this procedure can be used to compute iterated convolutions of sizes other than powers of 2; the details are left to the reader. The approximately Gaussian convolutions of Burt [3] can be obtained in this way using, e.g., iterated $4\times4$ convolutions; note that the prism machine yields the convolution values in every position, not just in sampled positions.

Burt [4] has shown that if we know the coefficients and error of the (least-squares) best-fitting polynomials of degree d to four contiguous $2^{k-1}\times2^{k-1}$ image blocks, we can directly compute the coefficients and error of the best-fitting polynomial of degree d to their union (a $2^k\times2^k$ block). His method can be implemented in a prism machine, by passing the information from the four subblocks to the level above, where the fit to the entire block is computed; note that this yields fits to $2^k\times2^k$ blocks in every position.

Other recent work on pyramid algorithms (e.g., Shneier [5]) involves fitting straight lines or other arcs to the edges or curves in each block of an image, and recursively combining these fits for groups of blocks whenever such combinations are possible. This too can be done in a prism machine by passing data fromthe subblocks to the level above; thus it too can be done for blocks in every position.

## 2.3. Representations at reduced scales

As we have just seen, a prism can carry out many of the same computations as a pyramid; but it cannot simulate a pyramid in every respect. In particular, it cannot represent information about an image at a reduced scale. When we use a prism to compute descriptions of $2^k \times 2^k$ blocks of an image, the descriptions of adjacent blocks are located $2^k$ apart on level k of the prism, whereas when we use a pyramid the descriptions of adjacent blocks are adjacent to one another. This adjacency may be advantageous; for example, we can use a pyramid to obtain condensed representations of the edges in an image, and we can then detect blob-like regions of arbitrary size in the image by discovering pixels (at appropriate levels of the pyramid) that are locally surrounded by edges [5]. With a prism, we would have to detect such regions by going to a higher level and looking for blocks in which the edge representations form a closed border, and this may be harder to do (the larger blocks will generally contain more edge information, and it may be harder to detect the desired combinations). Similar remarks apply to the detection of "ribbons" of arbitrary width in an image.

## 3. The discrete Fourier transform

Another advantage of a prism over a pyramid is that the prism can compute the discrete Fourier transform (DFT) of a $2^n \times 2^n$ input image in O(n) time. In this section we illustrate how this computation is carried out. We first show how to compute the one-dimensional DFT of a string of length $2^n$ in O(n) time on a one-dimensional "prism", consisting of n one-dimensional cellular arrays of length $2^n$ each, and where cell i on level k is joined to cells i and $i+2^k$ (modulo $2^n$) on level k+1.

We recall that the DFT of a string $z_0, z_1, \ldots, z_{N-1}$ of length N is defined by

$$w_r = \sum_{s=0}^{N-1} z_s e^{2\pi irs/N} \equiv \sum_{s=0}^{N-1} z_s E^{rs}, \text{ where } E \equiv e^{2\pi i/N}.$$

To see how these sums can be computed recursively, consider the case where N=8, and define the intermediate sequences $y_0, \ldots, y_7$ and $x_0, \ldots, x_7$ as shown in the first two columns below. It is then easily verified that the w's are as shown in the third column.

$$\begin{array}{lll}
y_0 = z_0+E^0 z_4 & x_0 = y_0+E^0 y_2 & w_0 = x_0+E^0 x_1 \\
y_1 = z_1+E^0 z_5 & x_1 = y_1+E^0 y_3 & w_1 = x_4+E^1 x_5 \\
y_2 = z_2+E^0 z_6 & x_2 = E^4 y_2+y_0 & w_2 = x_2+E^2 x_3 \\
y_3 = z_3+E^0 z_4 & x_3 = E^4 y_3+y_1 & w_3 = x_6+E^3 x_7 \\
y_4 = E^4 z_4+z_0 & x_4 = y_4+E^2 y_6 & w_4 = E^4 x_1+x_0 \\
y_5 = E^4 z_5+z_1 & x_5 = y_5+E^2 y_7 & w_5 = E^5 x_5+x_4 \\
y_6 = E^4 z_6+z_2 & x_6 = E^6 y_6+y_4 & w_6 = E^6 x_3+x_2 \\
y_7 = E^4 z_7+z_3 & x_7 = E^6 y_7+y_5 & w_7 = E^7 x_7+x_6
\end{array}$$

(This is a standard DFT algorithm.) We can compute these sequences in a three-level stack (of strings of length 8) as follows: Input the z's into level 3. Using the distance-4 connections between level 3 and level 2, compute the y's on level 2. (Each of the weighted sums can be computed in a single step, because the distance-4 connections are modulo 8.) Similarly, using the distance-2 connections between level 2 and level 1, compute the x's on level 1. (Here $x_0, x_1, x_4$, and $x_5$ can be computed in one step; but to get $x_2$, we must pass $y_0$ down to position 0 on level 1, pass it back up to position 2 on level 2 and combine it with $y_2$, and finally pass the result down to position 2 on level 1, and similarly for $x_3, x_6$, and $x_7$. We could eliminate the extra steps by connecting cell i on level k to cells i and $i \pm 2^k$ on level k+1.) Finally, by exchanging information between pairs of neighbors (0 and 1, 2 and 3,4 and 5, 6 and 7), compute the w's on level 1. Note that $w_0, w_2, w_5$, and $w_7$ end up in the right positions, but $w_1, w_3, w_4$ and $w_6$ end up in positions 4,6,1, and 3, respectively. We can switch them into the right positions by passing them back up through the stack. For example, we can move $w_1$ and $w_3$ from positions 4 and 6 on level 1 to positions 6 and 0 (=8 modulo 8) on level 2, then to positions 2 (=10 modulo 8) and 4 on level 3, and finally shift then one position to the left so they end up in positions 1 and 3. Similarly, we can move $w_4$ and $w_6$ from positions 1 and 3 on level 1 to positions 3 and 5 on level 2, then move them vertically upward to level 3 and shift them one position to the right so they end up in positions 4 and 6. At the same time, we can shift $w_0, w_2, w_5$ and $w_7$ vertically upward to the same positions on level 3. This example generalizes readily to a string of length $N=2^n$. The number of computational steps required is evidently $O(n)$.

To compute a two-dimensional DFT, we first use the process just described to compute the one-dimensional DFT of each row; note that it ends up where it began, in the top level. We now apply an exactly analogous process to this new input to compute the one-dimensional DFT of each column. (The row-wise computation used the connections between $(i,j)$ on level $k$ and $(i+2^k,j)$ on level $k+1$; the columnwise computation uses the connections between $(i,j)$ and $(i,j+2^k)$.) The final result is just the DFT of the input image. Note that by filtering this DFT (i.e., multiplying each $w$ value by some given weight) and then performing the inverse DFT, we can modify the original image in a variety of ways.

## 4. Prisms and hypercubes

In this section we briefly discuss the relationship between a prism and a hypercube of dimension 2n. For simplicity, we first show how a one-dimensional "prism," consisting of a stack of n one-dimensional cellular arrays of length $2^n$ each, is related to a hypercube of dimension n.

Imagine that we have an n-dimensional hypercube with one vertex located at the origin (i.e., at coordinates (0,...,0)) and with edges parallel to the axes. Thus the coordinates of any vertex are of the form $(\delta_1,\ldots,\delta_n)$, where each $\delta_i$ is 0 or 1. Moreover, each vertex is joined by edges to exactly n vertices $(\bar{\delta}_1,\delta_2,\ldots,\delta_n),(\delta_1,\bar{\delta}_2,\ldots,\delta_n),\ldots,(\delta_1,\delta_2,\ldots,\bar{\delta}_n)$, where $\bar{\delta}_i=1$ if $\delta_1=0$ and vice versa.

Now imagine that we have a string of cells of lengths $2^n$ in which each cell i $(i=0,1,\ldots,2^{n-1})$ is joined by an arc to the 2n cells $i\pm1,i\pm2,i\pm4,\ldots,i\pm2^{n-1}$. (Here addition is modulo $2^n$, i.e., if $i+2^k>2^n$, it becomes $i+2^k-2^n$, and if $i-2^k<0$, it becomes $i-2^k+_-{}^n$.) Since $0\le i<2^n$, i is an n-bit binary number, call it $\delta_1\ldots\delta_n$. If we identify i with the hypercube vertex $(\delta_1,\ldots,\delta_n)$, we see that the cells to which i is joined by arcs include all of the hypercube vertices to which $(\delta_1,\ldots,\delta_n)$ is joined by edges.

Finally, note that in a one-dimensional "prism", cell i on level k is joined to cells i and $i+2^k$ on level k+1 (so that cell i on level k+1 is joined to cells i and $i-2^k$ on level k). In other words, the 2n connections to cells $i\pm1,i\pm2,i\pm4,\ldots,i\pm2^n$ have been divided among the n levels.

This discussion generalizes straightforwardly to a two-dimensional prism and a hypercube of dimension 2n. In this hypercube, each vertex has coordinates $(\delta_1,\ldots,\delta_{2n})$, where each $\delta_i$ is 0 or 1. Thus we can identify each vertex with a cell (i,j) in a $2^n \times 2^n$ array, where the binary number representation of i is $\delta_1 \ldots \delta_n$ and that of j is $\delta_{n+1} \ldots \delta_{2n}$. We can obtain all the connections between hypercube vertices if we join each cell (i,j) by an arc to the 4n cells $(i\pm1,j),(i\pm2,j),(i\pm4,j),\ldots,(i\pm2^{n-1},j),(i,j\pm1),(i,j\pm2),(i,j\pm4),\ldots,(i,j\pm2^{n-1})$. In a prism, we have divided these 4n connections among the n levels.

The prism machine algorithms described in this paper could be easily modified to run on a 2n-dimensional hypercube, which would require only $4^n$ cells rather than $n \cdot 4^n$. However, in the hypercube each cell would have 2n neighbors, rather than 10 neighbors (4 on its own level, 3 on the level above, 3 on the level below) as in the prism. In particular, the between-level connections in the prism should be easier to implement than the connections in the hypercube.

## 5. Concluding remarks

We have defined a prism machine as a stack of n cellular arrays, each of size $2^n \times 2^n$, with cell (i,j) on level k connected to cells (i,j), $(i+2^k,j)$, and $(i,j+2^k)$ (modulo $2^n$) on level k+1. We have shown that such a machine can perform many useful types of operations on a $2^n \times 2^n$ image in O(n) time. These include histogramming; the discrete Fourier transform; and various types of convolution and polynomial fitting operations having kernels of sizes $2^k, k=1,2,\ldots,n$. These latter operations are performed in every position, rather than in positions spaced $2^k$ apart as in the case of a pyramid.*

The connections in a prism resemble those in a hypercube, except that we allow only connections in two "directions" at a time. The prism requires n times as many cells as a hypercube, but each cell has only a bounded number of neighbors instead of the O(n) neighbors in a hypercube. We shall show elsewhere that the between-level connections in a prism have a simple optical implementation.

The prism requires more cells than a hypercube or pyramid, but in practical cases the increase would be less than an order of magnitude. On the other hand, the prism has a very simple interconnection structure in which each cell has only a small number of neighbors, and in particular there are only three connections (per cell) between levels. Thus the prism deserves serious consideration as a possible architecture for image processing and analysis.

*It should be pointed out that pyramids can be defined to taper at arbitrary rates and to allow arbitrary degrees of overlap between the "sons" of neighboring nodes [3]. However, in all of these pyramid models the connections between each consecutive pair of levels are the same, whereas in our prism model the connections become increasingly spread out at high levels.

## References

1. S. Tanimoto and A. Klinger, eds., Structured Computer Vision: Machine Perception through Hierarchical Computation Structures, Academic Press, NY, 1980.

2. A. Rosenfeld, ed., Multiresolution Image Processing and Analysis, Springer, Berlin, 1984.

3. P. Burt, The pyramid as a structure for efficient computation, in [2], pp. 6-35.

4. P. Burt, Hierarchically derived piecewise polynomial approximations to waveforms and images, TR-838, Computer Vision Laboratory, University of Maryland, College Park, MD, November 1979.

5. M. Shneier, Multiresolution feature encodings, in [2], pp. 190-199.